Exercício Programa I : Decomposição LU para matrizes tridiagonais

MAP3121 - Métodos Numéricos e Aplicações

Prof. Dr. Pedro Peixoto

1º de maio, 2022

Laura do Prado Gonçalves Pinto
nºUSP 11819960

Mateus Stano Junqueira
nºUSP 11804845

Lista de Figuras

Conteúdo

1 Introdução

Neste relatório é apresentado todo o processo de construção lógica e computacional da resolução do primeiro exercício programa proposto para a disciplina MAP3121- Métodos Numéricos e Aplicações assim como os resultados obtidos pelos métodos apresentados e as conclusões tomadas sobre a tarefa proposta.

No exercício proposto trabalha-se a implementação de um algorítimo para decomposição de uma matriz tridiagonal pelo método LU, armazenando tanto a matriz inicial quanto as suas fatoradas em vetores. De posse de tal ferramenta , dedica-se em seguida à resolução de um sistema linear tridiagonal conforme as técnicas de decomposição LU mencionadas, utilizando os conhecimentos sobre o tópico.

Este relatório consta a metodologia utilizada para se alcançar tais objetivos além de todos os recursos utilizados para a construção do algoritmo assim como as referentes análises sobre os resultados encontrados.

2 Descrição

2.1 Análise do problema

O problema proposto na tarefa deste primeiro exercício programa consiste em utilizar um método de álgebra linear alternativo ao método de eliminação de Gauss conhecido como decomposição LU - que será explicado nos tópicos seguintes- para resolver sistemas lineares com características específicas quanto ao seu comportamento nas diagonais.

Em suma, o problema consiste em utilizar o método LU para resolver um sistema linear tridiagonal (cujas diagonais principal e secundárias superior e inferior não apresentam coeficientes nulos) cíclico.

Para obtenção deste tipo de algoritmo numérico , devemos inicialmente compreender os métodos e conceitos matemáticos envolvidos na solução analítica e, posteriormente compreender em como transferi-los para o código final para diferentes funções que podem ser combinadas única ou conjuntamente para resolução de diversas entradas de sistemas deste gênero.

2.2 Decomposição LU

Em álgebra linear, chama-se de decomposição LU o método de fatoração de uma matriz não singular como produto entre duas matrizes triangulares - uma superior (U) e outra inferior (L).

Trata-se de um método exato, ou seja, que fornece uma solução exata para o problema com poucos erros de arredondamento, à exceção da representação finita da máquina, e por meio de um número finito de operações.

Assim, a partir de A podemos obter os coeficientes de L e U sem necessariamente utilizar o método de eliminação de Gauss, de tal forma que

$$\begin{cases} u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, & i \leq j \\ l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj})/u_{jj}, & i > j \end{cases} \tag{1}$$

Como sabe-se que é possível obter o resultado para um sistema linear da forma $Ax = b$ de tal forma que ele seja decomposto em dois sistemas lineares triangular inferior e superior de forma que

$$\begin{cases} Ly = b \\ Ux = y \end{cases} \tag{2}$$

Assim, resolvendo a primeira equação obtemos y com o qual, substituindo na segunda equação obtemos x.

2.3 Matrizes tridiagonais e sistemas cíclicos

Matrizes tridiagonais são matrizes que possuem elementos não nulos apenas da diagonal principal e nas diagonais secundárias inferior e superior. Nestes casos específicos , é possível obter a decomposição LU de uma forma mais eficiente.

Sendo A uma matriz tridiagonal triangularizável pelo método de Gauss, sem trocar linhas os únicos elementos da matriz U que não podem ser nulos são U_{ii} e $U_{i,i+1}$ e os multiplicadores que não podem ser nulos são do tipo $L_{i+1,i}$. De tal forma que as operações aritméticas envolvidas para a resolução é reduzido de maneira considerável, dispensando cálculos de resultados que já conhecemos como nulos.

Assim, o armazenamento pode também ser mais eficiente, guardando apenas os valores que sabemos como não nulos por meio de três vetores chamados de a, b e c , os quais representam, respectivamente a diagonal secundária inferior, a diagonal principal e a diagonal secundária superior: $a = (0, a_2, ..., a_n); b = (b_1, b_2, ..., b_n); c = (c_1, ..., c_{n-1}, 0)$.

Assim, podemos também obter os coeficientes $u_i = U_{ii}$ e $l_{i+1} = L_{i+1,i}$ para depois obter L e U dos vetores armazenados destes elementos. Resolvendo então a equação $A \cdot x = d$.

Quando tratamos de problemas com periodicidade, é possível lidar com sistemas lineares tridiagonais cíclicos com soluções de decomposição LU. A soluções do sistema linear podem ser escritas como:

$$\begin{cases} T\tilde{x} + x_n v = \tilde{d} \\ w^t \tilde{x} + x_n b_n = d_n \end{cases} \tag{3}$$

Na qual T é a submatriz principal de A , de ordem $n-1$ e v e w são os vetores definidos por $v = (a_1, 0, ..., c_{n-1})^t$ e $w = (c_n, 0, ..., a_n)^t$ seguindo pelos vetores $\tilde{x} = (x_1, ..., x_{n-1})^t$ e $\tilde{d} = (d_1, ..., d_{n-1})$. Chegando na seguinte soluções :

$$x_n = \frac{d_n - c_n \tilde{y}_1 - a_n \tilde{y}_{n-1}}{b_n - c_n \tilde{z}_1 - a_n \tilde{z}_{n-1}} \quad \text{e} \quad \tilde{x} = \tilde{y} - x_n \tilde{z} \tag{4}$$

Na qual $\tilde{y}$ é a solução do sistema linar $T\tilde{y} = \tilde{d}$ e $\tilde{z}$ é a solução do sistema linear $T\tilde{z} = v$, ambos com a mesma matriz tridiagonal T de ordem $n - 1$.

3 Implementação do Código

3.1 Algoritmo

O algoritmo desejado, descrito no enunciado do exercício programa, pede que se implemente um método para decomposição LU de uma matriz tridiagonal n x n. Em seguida deve-se armazenar as matrizes A, L e U nos vetores descritos na explicações sobre sistemas tridiagonais.

Além disto, é demandado no exercício que sejam criadas funções para a resolução de sistema lineares tridiagonais, incluindo cíclicos até que por fim esses algoritmos sejam testados na resolução de um sistema tridiagonal cíclico cujos coeficientes de A, d e a ordem da matriz são dados no enunciado, colocado logo abaixo.

Tarefa

Implemente o algoritmo descrito acima para a decomposição LU de uma matriz tridiagonal A $n \times n$. As matrizes A, L e U devem ser armazenadas em vetores conforme descrito no texto. Implemente também o algoritmo para a resolução de um sistema linear tridiagonal usando a decomposição LU da matriz. *Faça as implementações de forma que elas possam ser usadas como partes de outros programas.*

Figura 1: Enunciado Exercício Programa I - primeira parte

Teste os algoritmos na resolução do sistema linear tridiagonal cíclico $Ax = d$, onde os coeficientes da matriz A são

$$a_i = \frac{2i-1}{4i},\ 1 \le i \le n-1,\ a_n = \frac{2n-1}{2n},$$
$$c_i = 1 - a_i,\ 1 \le i \le n,$$
$$b_i = 2,\ 1 \le i \le n,$$

e o lado direito do sistema linear é dado por

$$d_i = \cos\left(\frac{2\pi i^2}{n^2}\right),\ 1 \le i \le n.$$

Use $n = 20$. Pense em uma forma adequada para o programa apresentar os dados e as respostas.

Figura 2: Enunciado Exercício Programa I - parte dois

3.2 Metodologia aplicada

O algorítimo aplicado para a resolução desse exercício programa segue a seguinte linha de raciocínio:

- Em primeiro cria-se uma função capaz de efetuar a decomposição LU de qualquer matriz quadrada
- Implementa-se um código para resolução de sistema tridiagonais utilizando a lógica da decomposição LU para este caso específico
- código para resolução de sistemas tridiagonais cíclicos com e sem o input dos vetores diagonais

A seguir será destrinchado cada um destes seguimentos de raciocínio, numerados em modos I, II, III e IV, detalhando as funções, ferramentas e lógicas envolvidas em seu desenvolvimento assim como os resultados obtidos em cada um.

3.3 Modo I

O assim chamado "Modo I"neste exercício é o código dedicado a efetuar a decomposição LU de uma matriz n x n quadrada qualquer.

Para executar esse cálculo, criou-se uma função **def LU(A)** apresentada abaixo:

```
1  # Decompõe uma matriz quadrada de dimensão n na for A = LU
2  def LU(A):
3      # Dimensão n da matriz A
4      n = A.shape[0]
5
6      # Inicializa matrizes L e U
7      L = np.identity(n, dtype=np.float64)
8      U = np.zeros((n, n), dtype=np.float64)
9
10     # Calcula U e L
11     for i in range(n):
12         U[i, i:] = A[i, i:] - L[i, :i] @ U[:i, i:]
13         L[(i + 1) :, i] = (A[(i + 1) :, i] - L[(i + 1) :, :] @ U[:,
       i]) / U[i, i]
14
15     return U, L
```

O funcionamento desse modo é relativamente simples e segue os comentários. Em primeiro lugar se define a dimensão n da matriz quadrada qualquer A, utilizando o comando *.shape* como apresentado na sexta linha do código.

Em seguida, inicializa-se as matrizes fatoradas L e U, usando comandos da biblioteca *NumPy*, declarando o *dtype* como *float64* para minimizar erros de truncamento para melhor precisão numérica. Após isso, finalmente calcula-se as matrizes propriamente ditas criando um laço com as equações para obtenção dos termos de l_{ij} e u_{ij} apontadas no tópico 2.2 deste relatório.

Como mencionado no enunciado do exercício programa, para implementação dessas equações em *Python* é necessário somente o laço em i, e as equações dadas também são vetorizadas na operação de multiplicação de matrizes utilizando mais uma vez a *NumPy*.

3.4 Modo II

Neste modo trabalha-se a resolução de um sistema linear tridiagonal utilizando o método LU de decomposição porém de uma forma mais específica para matrizes com essa característica. Portanto, a lógica seguida envolve duas partes.

Na primeira parte, cria-se uma nova função capaz de decompor a matriz A em L e U, porém, com um método levemente diferente que o do modo um, otimizado para matrizes tridiagonais, como segue em anexo:

```
1  # Decompõe uma matriz tridiagonal quadrada de dimensão n na for A =
       LU
2  def decompoeMatrizTridiag(A):
3      # Dimensão n da matriz A
4      n = A.shape[0]
5
6      # Define os vetores das diagonais
7
```

```
    # Diagonal secundaria abaixo da principal
    a = [A[i + 1][i] for i in range(n - 1)]
    a = [0] + a  # Adiciona 0 a primeiro item do vetor

    # Diagonal principal
    b = [A[i][i] for i in range(n)]

    # Diagonal secundaria acima da principal
    c = [A[i][i + 1] for i in range(n - 1)]
    c = c + [0]  # Adiciona 0 ao fim do vetor

    return a, b, c
```

Para esta função **def decompoeMatrizTridiag**, inicia-se da mesma forma: definindo a dimensão de A. Porém, para sua decomposição, criam-se laços para definição de seus vetores diagonais (a, b e c) que correspondem respectivamente ao vetor da diagonal secundária abaixo da principal, o vetor da diagonal principal e o vetor da diagonal secundária acima da principal. A obtenção de todos eles vem da lógica exemplificada no item 2.3 deste relatório, em que basicamente se extraem os coeficientes desejados da matriz A a ser decomposta.

Um detalhe a ser acrescentado é que para obtenção dos vetores como descritos no enunciado do exercício programa, faz-se nas diagonais secundárias uma equação fora do laço para acrescentar um zero no início e no final dos vetores a e c respectivamente, por meio de uma simples soma de listas.

Seguindo para a segunda parte do modo, encontramos a necessidade de resolver o sistema linear tridiagonal utilizando os vetores encontrados, por meio de três laços consecutivos após a utilização da parte um como mostra o listing abaixo:

```
# Resolve sistema tridiagonais
def sistemaTridiagLU(A, d):
    # Dimensão de A
    n = A.shape[0]

    # Define diagonais a, b e c
    a, b, c = decompoeMatrizTridiag(A)

    # Calcula vetores u e l
    u = [b[0]]
    l = []
    for i in range(1, n):
        l.append(a[i] / u[i - 1])
        u.append(b[i] - l[i - 1] * c[i - 1])

    # Calcula solução de L*y = d
    y = [d[0]]
    for i in range(1, n):
        y.append(d[i] - l[i - 1] * y[i - 1])

    # Calcula solução de U*x = y
    x = [0] * n
    x[n - 1] = y[n - 1] / u[n - 1]
```

```
    for i in reversed(range(0, n - 1)):
        x[i] = (y[i] - c[i] * x[i + 1]) / u[i]

    return x
```

Em primeiro lugar, cria-se um laço para a construção dos vetores u e l, coeficientes das matriz L e U, seguindo as instruções do algoritmo exemplificado no enunciado do exercício programa para em seguida implementar o vetor l para resolução de $L \cdot y = d$ de forma a obter y.

Desta forma, em posse da lista do vetor y, seguimos para o último laço, do algoritmo também demonstrado no enunciado, para resolução de $U \cdot x = y$ de forma a finalmente obter o vetor x do sistema linear formado por $A \cdot x = d$.

3.5 Modo III

No último modo composto para este exercício programa, trabalha-se a resolução sistema tridiagonais cíclicos ainda utilizando ferramentas da decomposição LU porém focadas na resolução desse gênero de sistema linear. Também é possível dividir esse algoritmo em três partes.

A primeira parte, pertencente a *main* do código, onde define-se os vetores diagonais a, b, c e vetor d, conforme as equações estabelecidas no enunciado da tarefa.

```
elif modo == 3:
        n = int(input("Dimensão da matriz tridiagonal: "))

        # Inicializa listas
        a = np.zeros(n, dtype=np.float64)
        b = np.zeros(n, dtype=np.float64)
        c = np.zeros(n, dtype=np.float64)
        d = np.zeros(n, dtype=np.float64)

        # Calcula a, b, c e d
        for i in range(n):
            a[i] = (
                (2 * (i + 1) - 1) / (4 * (i + 1))
                if (i + 1) != n
                else (2 * (i + 1) - 1) / (2 * (i + 1))
            )
            c[i] = 1 - a[i]
            b[i] = 2
            d[i] = np.cos((2 * np.pi * (i + 1) ** 2) / (n**2))

        x, A = sistemaTridiagCiclico(a, b, c, d)

        print("\nVetor a: ")
        print(a)
        print("\nVetor b: ")
        print(b)
        print("\nVetor c: ")
        print(c)
        print("\nVetor d: ")
```

```
30          print(d)
31          print("\nSolução x: ")
32          for i in x:
33              print(i)
```

A segunda parte, descrita no listing abaixo, diz respeito a construção da matriz principal A a partir dos vetores a ,b, e c, mencionados no modo anterior . Do vetor b , retiramos a dimensão da matriz A - pelo comprimento da lista representante deste vetor. Assim, é possível repetir a função que inicializa a matriz, como no modo I e criar um laço subsequente no qual se utiliza dos três vetores diagonais para reconstruir a matriz A.

```
1 # Cria matriz tridiagonal ciclica a partir dos vetores que definem
    suas diagonais
2 def criaMatrizTridiagCiclico(a, b, c):
3     # Dimensão n da matriz
4     n = len(b)
5
6     # Inicializa matriz de dimensão n
7     A = np.zeros((n, n), dtype=np.float64)
8
9     # Constrõe a matriz a partir de a,b e c
10    for i in range(n):
11        if i == 0:
12            A[i][n - 1] = a[i]
13        else:
14            A[i][i - 1] = a[i]
15        A[i][i] = b[i]
16        if i == n - 1:
17            A[i][0] = c[i]
18        else:
19            A[i][i + 1] = c[i]
20
21    return A
```

A última parte é o algoritmo reservado para a resolução do sistema tridiagonal cíclico, estabelecendo a função **def sistemaTridiagCiclico**. Nesta função, novamente se estabelece a grandeza da matriz pelo comprimento do vetor b e após tal constrói-se a matriz A a partir de seus vetores diagonais.

Então, com a matriz A montada, é a vez de criar a submatriz principal T, como descrita no item 2.3 deste relatório, utilizando as equações referidas tanto neste mesmo tópico desse relatório e no enunciado do exercício.

Em seguida, dedica-se a construir o vetor v e, posteriormente em resolver as equações $T \cdot \tilde{y} = d$ e $T \cdot \tilde{z} = v$. Com esses dados em mãos, segue-se para o cálculo de x_n e x baseados nas equações apontadas no enunciado do exercício.

```
1 # Resolve sistemas tridiagonais ciclicos
2 def sistemaTridiagCiclico(a, b, c, d):
3     n = len(b)
4
5     # Constroi matriz A a partir de a,b e c
6     A = criaMatrizTridiagCiclico(a, b, c)
7
```

```
8       # Constroi submatriz principal T
9       T = np.delete(A, n - 1, 1)
10      T = np.delete(T, n - 1, 0)
11
12      # Constroi v
13      v = [0] * n
14      v[0] = a[0]
15      v[-1] = c[n - 2]
16
17      # Resolve sistema Ty=d
18      y = sistemaTridiagLU(T, d)
19
20      # Resolve sistema Tz=v
21      z = sistemaTridiagLU(T, v)
22
23      # Solução do sistema
24      x = [0] * n
25
26      # Calcula xn
27      x[n - 1] = (d[(n - 1)] - c[(n - 1)] * y[0] - a[(n - 1)] * y[(n -
        1) - 1]) / (
28          b[(n - 1)] - c[(n - 1)] * z[0] - a[(n - 1)] * z[(n - 1) - 1]
29      )
30      # Calcula os valores de x restantes
31      for i in range(n - 1):
32          x[i] = y[i] - x[n - 1] * z[i]
33
34      return x, A
```

3.6 Modo IV

Neste último modo, utilizamos o mesmo modo de resolução de sistema linear tridiagonal cíclico utilizado no modo anterior, contudo não obtemos os vetores diagonais e o vetor de d de nenhuma matriz - no caso, eles são um input colocado pelo usuário.

4 Conclusão

4.1 Resultados

Seguem abaixo os resultados e comentários sobre os devidos outputs testados para verificação dos códigos apresentados neste exercício programa.

4.1.1 Modo I

Para a verificação deste modo a estratégia utilizada foi obter os resultados para determinada matriz primeiro pelo código em *Python* e em seguida colocar a mesma matriz em uma calculadora online que se propõe a disponibilizar o mesmo resultado.

As imagens a seguir demonstram que os resultados obtidos por ambos os meios são iguais para se afirmar ter confiança na funcionalidade do código proposto.

```
Matriz de entrada:
[[1. 1. 2. 1.]
 [3. 2. 1. 4.]
 [4. 2. 3. 4.]
 [5. 6. 7. 8.]]
Matriz L:
[[ 1.   0.   0.   0. ]
 [ 3.   1.   0.   0. ]
 [ 4.   2.   1.   0. ]
 [ 5.  -1.  -1.6  1. ]]
Matriz U:
[[ 1.   1.   2.   1. ]
 [ 0.  -1.  -5.   1. ]
 [ 0.   0.   5.  -2. ]
 [ 0.   0.   0.   0.8]]
```

Figura 3: Input e output teste para resolução do modo I

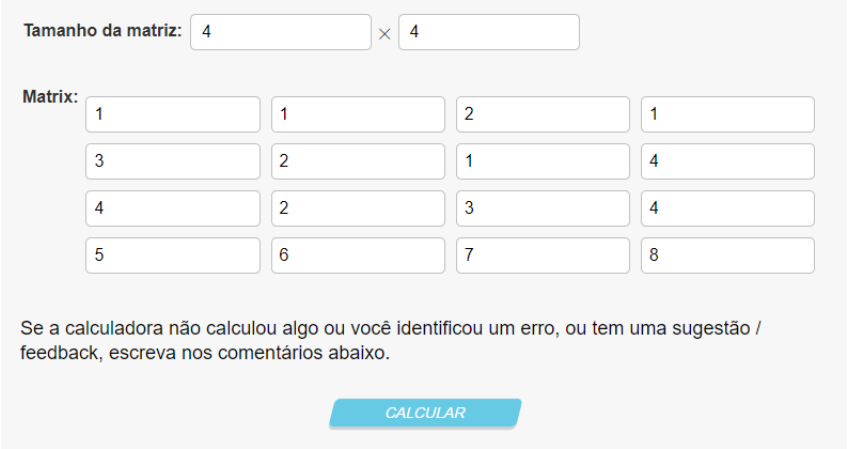

Figura 4: Input da mesma matriz em uma calculadora online

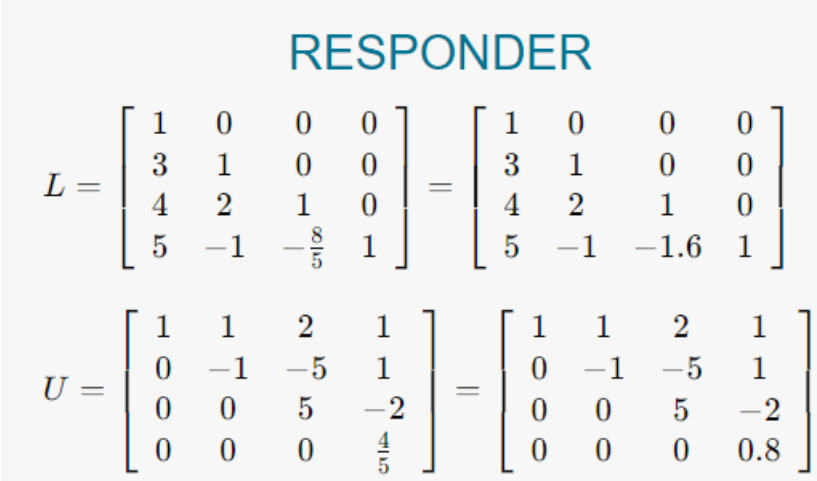

Figura 5: Output da referida matriz na calculadora online

4.1.2 Modo II

Para a verificação deste modo utilizou-se a mesma estratégia que para verificação do anterior: encontrou-se um dispositivo online que se propõe a obter a mesma resposta que o código elaborado em *Python*.

As imagens a seguir confirmam que os resultados obtidos para ambos os meios são suficiente para atestar a confiança no código elaborado.

```
Matriz de entrada:
[[5. 4. 0. 0. 0. 0.]
 [1. 3. 1. 0. 0. 0.]
 [0. 2. 4. 1. 0. 0.]
 [0. 0. 1. 2. 1. 0.]
 [0. 0. 0. 2. 3. 2.]
 [0. 0. 0. 0. 1. 2.]]
Insira itens do vetor y
y0: 1
y1: 2
y2: 3
y3: 4
y4: 5
y5: 6
Vetor y:
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
Solução x:
[-1.0, 1.5, -1.5000000000000002, 6.0, -6.499999999999999, 6.249999999999999]
```

Figura 6: Input e output teste para resolução do modo II

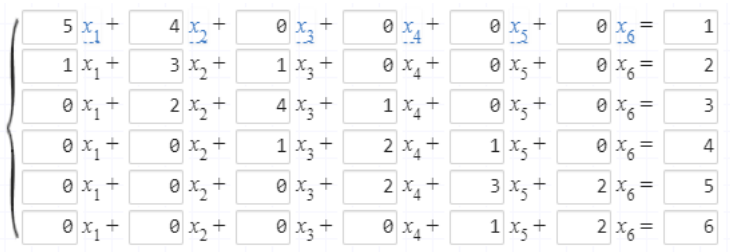

Figura 7: Input da mesma matriz em uma calculadora online

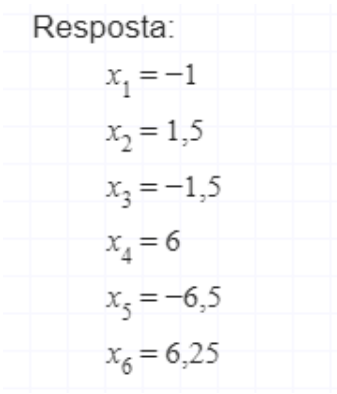

Figura 8: Output da referida matriz na calculadora online

4.1.3 Modo III

Após atestar a funcionalidade de ambos os modos anteriores os quais foram a base para a construção deste e do próximo método de resolução para os sistema linear proposto na tarefa deste exercício programa, finalmente obtem-se os seguintes resultados para o sistema proposto:

```
Vetor a:
[0.25       0.375      0.41666667 0.4375     0.45       0.45833333
 0.46428571 0.46875    0.47222222 0.475      0.47727273 0.47916667
 0.48076923 0.48214286 0.48333333 0.484375   0.48529412 0.48611111
 0.48684211 0.975     ]

Vetor b:
[2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]

Vetor c:
[0.75       0.625      0.58333333 0.5625     0.55       0.54166667
 0.53571429 0.53125    0.52777778 0.525      0.52272727 0.52083333
 0.51923077 0.51785714 0.51666667 0.515625   0.51470588 0.51388889
 0.51315789 0.025     ]

Vetor d:
[ 9.99876632e-01  9.98026728e-01  9.90023658e-01  9.68583161e-01
  9.23879533e-01  8.44327926e-01  7.18126298e-01  5.35826795e-01
  2.94040325e-01  6.12323400e-17 -3.23917418e-01 -6.37423990e-01
 -8.83765630e-01 -9.98026728e-01 -9.23879533e-01 -6.37423990e-01
 -1.71929100e-01  3.68124553e-01  8.18149717e-01  1.00000000e+00]
```

Figura 9: Output dos vetores diagonais e vetor d para o exercício proposto

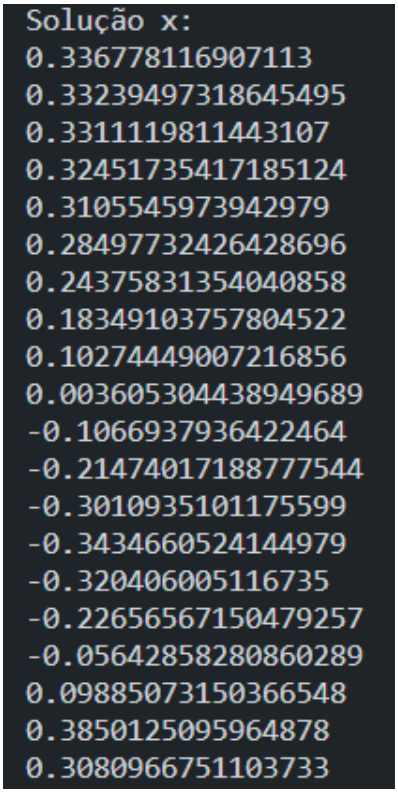

Figura 10: Output de soluções para x

Por conta da resolução em sistema cíclico não foi possível efetuar o mesmo nível de conferência da confiabilidade dos resultados que com as funções mais simples deste exercício, contudo os dois primeiro sucessos são significativos indicativos de boa execução das demais funções de resoluções as quais são semi-dependentes destes. A existência de pouca distorção entre os dados obtidos para os vetores e as soluções em x também são um bom indicativo de uma resolução coerente.

4.1.4 Modo IV

Para este último teste, foram utilizados métodos menos rigorosos de conferência, uma vez que usa ad mesmas funções que o modo anterior, com exceção da obtenção dos vetores a,b,c e d que neste caso são inputs.

É possível obter o exato mesmo resultado que no modo anterior por razões triviais de mesmo cálculo utilizado. Como existe a mesma deficiência de segundo teste que no modo anterior, buscou-se efetuar uma matriz mais simples e comparar a coerência entre os dados colocados e a matriz obtida:

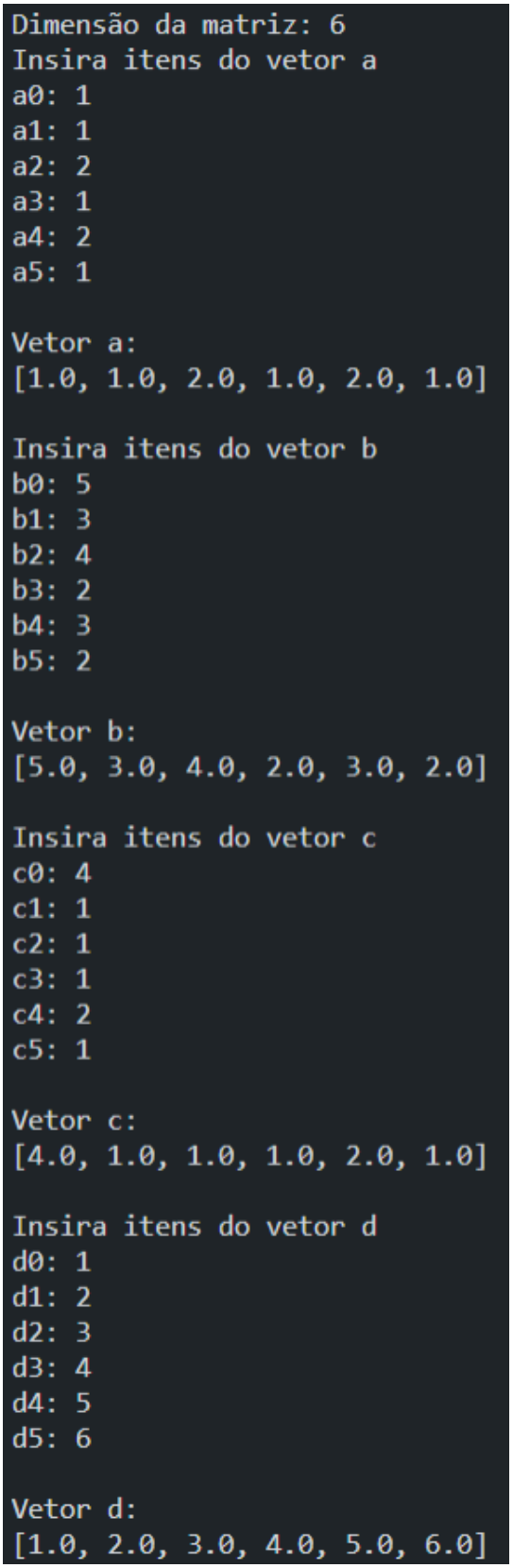

Figura 11: Input para os vetores diagonais e vetor d no modo IV

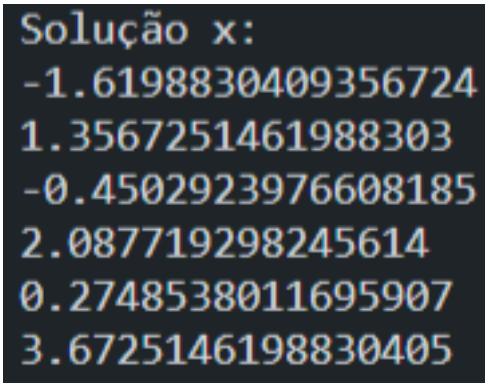

Figura 12: resultados em x para os inputs de matriz de ordem 6

5 Referências

JUSTINO. Lucas; **Método da Decomposição LU para a solução numérica de equações lineares**. Disponível em :

"http://www.facom.ufu.br/ dino/disciplinas/GBC051/Decomp_LU_Lucas.pdf"

ECT / UFRN. **Decomposição LU (Lower Upper)**. Disponível em: <https://cn.ect.ufrn.br/index.php?r=conteudo%2Fsislin-lu>. Acesso em: 1 maio 2022.

RISTO HINNO. LU decomposition - Risto Hinno - Medium. Medium. Disponível em: <https://ristohinno.medium.com/lu-decomposition-41a3cb0d1ba>. Acesso em: 1 maio 2022.

Matrix Calculator. Matrixcalc.org. Disponível em: <https://matrixcalc.org/pt/slu.html>. Acesso em: 2 maio 2022.

Calculadora de Decomposição LU - eMathHelp. Emathhelp.net. Disponível em: <https://www.emathhelp.net/pt/calculators/linear-algebra/lu-decomposition-calculator/>. Acesso em: 2 maio 2022.

NumPy documentation — NumPy v1.22 Manual. Numpy.org. Disponível em: <https://numpy.org/doc/stable/index.html>. Acesso em: 2 maio 2022.

A Apêndices

A.1 Main.py

```
import numpy as np

from functions import (
    LU,
    sistemaTridiagLU,
    sistemaTridiagCiclico,
)


def main():
    print("Escolha um modo:")
    print("Modo 1 - Faz decomposição LU de uma matriz nxn")
    print("Modo 2 - Resolve um sistema tridiagonal")
    print("Modo 3 - Resolve sistema tridiagonal ciclico com matriz pré definida")
    print("Modo 4 - Resolve sistema tridiagonal ciclico com matriz escolhida")
    modo = int(input("Modo: "))

    if modo == 1:
        n = int(input("Dimensão da Matriz: "))
        A = np.zeros((n, n), dtype=np.float64)
        for i in range(n):
            for j in range(n):
                a_ij = float(input("item a" + str(i) + str(j) + ": "))
                A[i][j] = a_ij

        print("Matriz de entrada: ")
        print(A)

        U, L = LU(A)

        print("Matriz A: ")
        print(A)
        print("Matriz L: ")
        print(L)
        print("Matriz U: ")
        print(U)

    elif modo == 2:
        n = int(input("Dimensão da Matriz do sistema: "))
        A = np.zeros((n, n), dtype=np.float64)
        for i in range(n):
            for j in range(n):
                a_ij = float(input("Item a" + str(i) + str(j) + ": "))
```

```
                A[i][j] = a_ij

        print("Matriz de entrada: ")
        print(A)

        print("Insira itens do vetor y")
        y = [0] * n
        for i in range(n):
            yi = float(input("y" + str(i) + ": "))
            y[i] = yi

        print("Vetor y: ")
        print(y)

        x = sistemaTridiagLU(A, y)

        print("Solução x: ")
        print(x)

    elif modo == 3:
        n = int(input("Dimensão da matriz tridiagonal: "))

        # Inicializa listas
        a = np.zeros(n, dtype=np.float64)
        b = np.zeros(n, dtype=np.float64)
        c = np.zeros(n, dtype=np.float64)
        d = np.zeros(n, dtype=np.float64)

        # Calcula a, b, c e d
        for i in range(n):
            a[i] = (
                (2 * (i + 1) - 1) / (4 * (i + 1))
                if (i + 1) != n
                else (2 * (i + 1) - 1) / (2 * (i + 1))
            )
            c[i] = 1 - a[i]
            b[i] = 2
            d[i] = np.cos((2 * np.pi * (i + 1) ** 2) / (n**2))

        x, A = sistemaTridiagCiclico(a, b, c, d)

        print("\nVetor a: ")
        print(a)
        print("\nVetor b: ")
        print(b)
        print("\nVetor c: ")
        print(c)
        print("\nVetor d: ")
        print(d)
        print("\nSolução x: ")
        for i in x:
```

```
95              print(i)
96
97      elif modo == 4:
98          n = int(input("Dimensão da matriz: "))
99
100         print("Insira itens do vetor a")
101         a = [0] * n
102         for i in range(n):
103             ai = float(input("a" + str(i) + ": "))
104             a[i] = ai
105
106         print("\nVetor a: ")
107         print(a)
108
109         print("\nInsira itens do vetor b")
110         b = [0] * n
111         for i in range(n):
112             bi = float(input("b" + str(i) + ": "))
113             b[i] = bi
114
115         print("\nVetor b: ")
116         print(b)
117
118         print("\nInsira itens do vetor c")
119         c = [0] * n
120         for i in range(n):
121             ci = float(input("c" + str(i) + ": "))
122             c[i] = ci
123
124         print("\nVetor c: ")
125         print(c)
126
127         print("\nInsira itens do vetor d")
128         d = [0] * n
129         for i in range(n):
130             di = float(input("d" + str(i) + ": "))
131             d[i] = di
132
133         print("\nVetor d: ")
134         print(d)
135
136         x, A = sistemaTridiagCiclico(a, b, c, d)
137
138         print("\nSolução x: ")
139         for i in x:
140             print(i)
141
142     else:
143         raise Exception("Modo não existente")
144
145
```

```
146 if __name__ == "__main__":
147     main()
```

A.2 Functions.py

```
1  import numpy as np
2
3  # Decompõe uma matriz quadrada de dimensão n na for A = LU
4  def LU(A):
5      # Dimensão n da matriz A
6      n = A.shape[0]
7
8      # Inicializa matrizes L e U
9      L = np.identity(n, dtype=np.float64)
10     U = np.zeros((n, n), dtype=np.float64)
11
12     # Calcula U e L
13     for i in range(n):
14         U[i, i:] = A[i, i:] - L[i, :i] @ U[:i, i:]
15         L[(i + 1):, i] = (A[(i + 1):, i] - L[(i + 1):, :] @ U[:,
       i]) / U[i, i]
16
17     return U, L
18
19
20 # Decompõe uma matriz tridiagonal quadrada de dimensão n na for A =
       LU
21 def decompoeMatrizTridiag(A):
22     # Dimensão n da matriz A
23     n = A.shape[0]
24
25     # Define os vetores das diagonais
26
27     # Diagonal secundaria abaixo da principal
28     a = [A[i + 1][i] for i in range(n - 1)]
29     a = [0] + a  # Adiciona 0 a primeiro item do vetor
30
31     # Diagonal principal
32     b = [A[i][i] for i in range(n)]
33
34     # Diagonal secundaria acima da principal
35     c = [A[i][i + 1] for i in range(n - 1)]
36     c = c + [0]  # Adiciona 0 ao fim do vetor
37
38     return a, b, c
39
40
41 # Resolve sistema tridiagonais
42 def sistemaTridiagLU(A, d):
43     # Dimensão de A
```

```
44      n = A.shape[0]
45
46      # Define diagonais a, b e c
47      a, b, c = decompoeMatrizTridiag(A)
48
49      # Calcula vetores u e l
50      u = [b[0]]
51      l = []
52      for i in range(1, n):
53          l.append(a[i] / u[i - 1])
54          u.append(b[i] - l[i - 1] * c[i - 1])
55
56      # Calcula solução de L*y = d
57      y = [d[0]]
58      for i in range(1, n):
59          y.append(d[i] - l[i - 1] * y[i - 1])
60
61      # Calcula solução de U*x = y
62      x = [0] * n
63      x[n - 1] = y[n - 1] / u[n - 1]
64      for i in reversed(range(0, n - 1)):
65          x[i] = (y[i] - c[i] * x[i + 1]) / u[i]
66
67      return x
68
69
70  # Cria matriz tridiagonal ciclica a partir dos vetores que definem
        suas diagonais
71  def criaMatrizTridiagCiclico(a, b, c):
72      # Dimensão n da matriz
73      n = len(b)
74
75      # Inicializa matriz de dimensão n
76      A = np.zeros((n, n), dtype=np.float64)
77
78      # Constrõe a matriz a partir de a,b e c
79      for i in range(n):
80          if i == 0:
81              A[i][n - 1] = a[i]
82          else:
83              A[i][i - 1] = a[i]
84          A[i][i] = b[i]
85          if i == n - 1:
86              A[i][0] = c[i]
87          else:
88              A[i][i + 1] = c[i]
89
90      return A
91
92
93  # Resolve sistemas tridiagonais ciclicos
```

```
94  def sistemaTridiagCiclico(a, b, c, d):
95      n = len(b)
96
97      # Constroi matriz A a partir de a,b e c
98      A = criaMatrizTridiagCiclico(a, b, c)
99
100     # Constroi submatriz principal T
101     T = np.delete(A, n - 1, 1)
102     T = np.delete(T, n - 1, 0)
103
104     # Constroi v
105     v = [0] * n
106     v[0] = a[0]
107     v[-1] = c[n - 2]
108
109     # Resolve sistema Ty=d
110     y = sistemaTridiagLU(T, d)
111
112     # Resolve sistema Tz=v
113     z = sistemaTridiagLU(T, v)
114
115     # Solução do sistema
116     x = [0] * n
117
118     # Calcula xn
119     x[n - 1] = (d[(n - 1)] - c[(n - 1)] * y[0] - a[(n - 1)] * y[(n -
        1) - 1]) / (
120         b[(n - 1)] - c[(n - 1)] * z[0] - a[(n - 1)] * z[(n - 1) - 1]
121     )
122     # Calcula os valores de x restantes
123     for i in range(n - 1):
124         x[i] = y[i] - x[n - 1] * z[i]
125
126     return x, A
127
```